Naval Surface Warfare Center
Carderock Division
West Bethesda, MD 20817-5700

NSWCCD-CISD–2009/008 August 2009

Ship Systems Integration & Design Department

Technical Report

DUKW-21 Autonomous Navigation

Autonomous Path Planning for an Amphibious Vehicle

By

Benjamin Flom

NSWCCD-CISD-2009-008 DUKW-21 Autonomous Navigation

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
7-Aug-2009

2. REPORT TYPE
Final

3. DATES COVERED *(From - To)*
18-May-2009 – 07-August-2009

4. TITLE AND SUBTITLE
DUKW-21 Autonomous Navigation - Autonomous Path Planning for an Amphibious Vehicle

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Benjamin Flom

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

8. PERFORMING ORGANIZATION REPORT NUMBER
NSWCCD-CISD-2009/008

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Surface Warfare Center
Carderock Division
9500 MacArthur Boulevard
West Bethesda, MD 20817-5700

10. SPONSOR/MONITOR'S ACRONYM(S)

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Chief of Naval Research
One Liberty Center
875 North Randolph Street,
Suite 1425
Arlington, VA 22203-1995

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Approved for Public Release: Distribution Unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
Amphibious vehicles, which transport cargo from ship to shore, will play a critical role in future Sea Base supply chain operations. During the summer of 2007, a project team of CISD interns developed a design concept called DUKW-21, a manned amphibious vehicle that had autonomous compatibility so that in the future, an autonomous control system could be implemented. To start the design of an autonomous control system for DUKW-21, this 2009 CISD intern project focused on autonomous amphibious path planning. While there have been developments in unmanned vehicle navigation, they have been focused on operations on land, sea or air with no integration of operational modes. Unmanned amphibious vehicles face a unique challenge in autonomously transitioning between sea and land, where there has been little research. Existing research on ground and sea path optimization algorithms has been compiled, which would be a part of the overall amphibious navigation process. An original, intuitive, algorithm is proposed for transitioning, which makes up the final piece of the amphibious path planning algorithm. Several functional requirements are defined for DUKW-21 that are needed for implementation of the proposed algorithms.

15. SUBJECT TERMS
DUKW-21, DUKW-ling, path-planning, autonomous amphibious navigation, CISD

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Colen Kennell
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL	27	**19b. TELEPHONE NUMBER** *(include area code)* 301-227-5468

Abstract

Amphibious vehicles, which transport cargo from ship to shore, will play a critical role in future Sea Base supply chain operations. During the summer of 2007, a project team of CISD interns developed a design concept called DUKW-21, a manned amphibious vehicle that had autonomous compatibility so that in the future, an autonomous control system could be implemented.

To start the design of an autonomous control system for DUKW-21, this 2009 CISD intern project focused on autonomous amphibious path planning. While there have been developments in unmanned vehicle navigation, they have been focused on operations on land, sea or air with no integration of operational modes. Unmanned amphibious vehicles face a unique challenge in autonomously transitioning between sea and land, where there has been little research.

This report investigates autonomous navigation by compiling ground and sea path optimization algorithms, a part of the overall amphibious navigation process. It then proposes an original, intuitive, algorithm for transitioning, which bridges the gap between sea and land path planning. Several functional requirements are outlined for the DUKW-21 project that are needed for implementation of the proposed algorithms.

Acknowledgements

This report is the culmination of work conducted by a University of Maryland undergraduate student hired under the National Research Enterprise Intern Program sponsored by the Office of Naval Research. The work was performed in the Center for Innovation in Ship Design (CISD), which is part of the Ship Systems Integration and Design Department at Naval Surface Warfare Center Carderock Division. Acknowledged are several individuals that have been of great assistance in the development of this project:

- Chris Wilson, Steve Ouimette, Jack Offutt, and Dr. Colen Kennell of CISD.
- Special thanks to Prof. Irina Dolinskaya of Northwestern University.

Table of Contents

List of Figures

Nomenclature

AGV: Autonomous Ground Vehicle
ASSV: Autonomous Sea-Surface Vehicle
AuAV: Autonomous Amphibious Vehicle
AV: Autonomous Vehicle
CISD: Center for Innovation In Ship Design
LMSR: Large Medium-Speed Roll on/ Role off ship
SWATH: Small Waterplane Area Twin Hull
ULPAR: Unit Linear Path Attainable Region

Introduction

The United States military currently relies heavily on large transport vessels, such as Large Medium-Speed RoRo (LMSR) ships, and developed deep water port facilities to deliver logistic materiel abroad. The difficulties in finding and securing port facilities in hostile areas spurred the development of the Sea Base concept; the provision of mobile port facilities in controlled waters. The key challenge to the concept is the development of a supply chain to transport cargo from the Sea Base to shore. One approach is to use a medium size container ship to transport cargo from the Sea Base to a location close to shore, at which point amphibious vehicles can run continuous delivery missions.

During the summer of 2007, a project team of CISD interns developed a design concept called DUKW-21, an amphibious vehicle that would enhance sea to shore logistics (Gonzalez *et al.*, see **Error! Reference source not found.**). The design's simplicity allows it to drive over, pick up, and carry a 20ft ISO container between its SWATH-like hulls from a cargo ship five nautical miles offshore to a point five nautical miles inland.

As a continuation of the concept design, a number of prototype models have been produced, the latest of which is a 1:7 scale remote-control model called DUKW-ling (Critchell, see **Error! Reference source not found.**). Its purpose is to demonstrate the navigational capabilities of the original DUKW-21 concept, primarily its ability to operate both in and out of water, and transition between water and land.

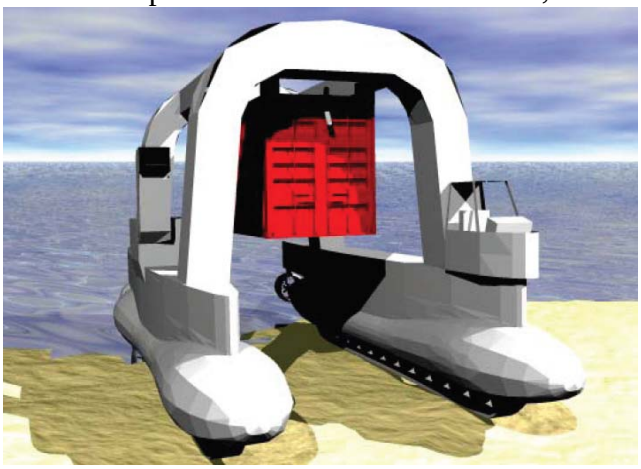

Figure 1: CAD Rendering of DUKW-21 Concept

Figure 2: DUKW-ling Demonstrator

DUKW-21 Automation

One of the initial requirements of the DUKW-21 concept was to be "controlled by either a single crew member or by automatic, unmanned control," allowing a fleet of DUKWs to continuously facilitate the ship-to-shore logistics train with limited human interaction. Replacing the human driver with an intelligent computer would remove the risk of

casualty, and it would allow the vehicle to make decisions in real-time, even if communication was temporarily interrupted.

The purpose of this project is to investigate possible autonomous control philosophies and propose an autonomous navigation algorithm that could be used to travel from a point offshore to a point inland and back. A number of assumptions have been made regarding the type of information DUKW-21 will need in order to execute its mission, and they are outlined in the conclusion of this report.

Overview of Autonomous Navigation

Autonomy is the capacity of a system to make informed, un-coerced decisions about its actions without the involvement of another system or operator. In recent years it has been adopted by the fields of robotics and manufacturing to describe systems or products which perform complex, potentially hazardous, repetitive or mundane activities with only minor human supervision or instruction.

Autonomous vehicles (AV), be they (sea) surface, ground or air based, generally consist of the same four core components, as illustrated in Figure 33:

- A perception interface, which consists of sensors that acquire information about the system's environment, as well as software that converts low-level input signals from the sensors into high-level information.
- A planner, which, based on the information acquired by the perception interface, as well as knowledge about the system's present state, produces the best high-level plan for the system to complete its mission.
- An executive, which upon reading a new plan, calculates what the actuators need to do for the system to run the plan, and outputs high-level commands to the actuator interface.
- An actuator interface, which consists of moveable components, as well as software that converts high-level commands into low-level signals that control the motion of the actuators.

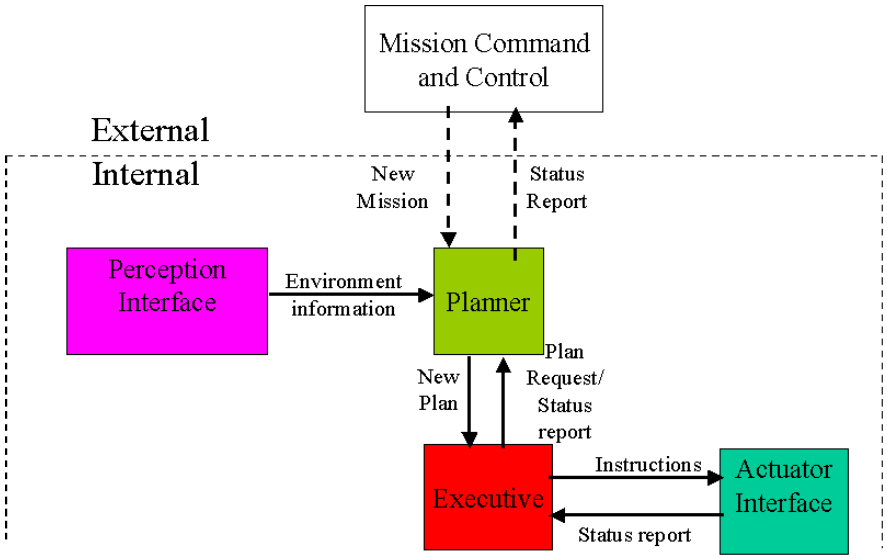

Figure 3: Autonomous Control Architecture

An AV planner would derive plans that include vehicle navigation, as well as other movements (such as picking up and transferring an ISO container, in the case of DUKW-21). The process of producing a navigation plan is called *path planning*. The scope of this project is limited to path planning for an Autonomous Amphibious Vehicle (AuAV).

Path Planning Philosophies

Amphibious vessels are unique in that they operate in three distinct zones; on the surface of water, on land and (at various stages) in both at the same time. Current developments in vehicle autonomy have focused on operations on land, sea or air with no combinations. Although an amphibious vessel could take advantage of these ideas, discreet philosophies have been developed in each area to take advantage of their individually unique problems.

Autonomous Ground Vehicles (AGV) typically implement a batch path planning system, which finds a *complete* path from the present location to the goal waypoint (Pell *et al.*). A path is said to be complete if the vehicle can successfully arrive at the goal destination by following the path (assuming the environment does not change). Batch planners work well with AGVs because ground environments are mostly static (for example, terrain elevations do not change by the minute), and if there are only minor changes, the path does not need to be re-planned from scratch because most of it remains unaffected (Ferguson *et al.*).

Autonomous Sea Surface Vehicles (ASSV) do not typically implement batch path algorithms because the sea environment is very dynamic; the path would often need to be completely re-calculated, resulting in extensive computation time. Instead, ASSVs usually implement a continuous path planner, which finds an optimal path to a horizon point, well short of the goal waypoint. This plan is continuously updated with modifications as the vehicle moves (Huntsberger *et al.*, Chien *et al.*, Larson *et al.*).

Autonomous Amphibious Vehicles (AuAV) also face a particularly difficult challenge: the *transition*. That is, traveling through both environments while moving from sea to land, and vice versa. This transition requires the vehicle to traverse the surf zone, which is even more dynamic than open water, negating the use of a batch path planner. Similarly, a continuous path planner which does not consider the entire environment would also fail, for the same reason it is unsuitable for ground navigation: it only plans for the short term, potentially allowing the AV to get stuck in a dead end. See Figure 4 for an illustration of this problem. Figure 4 illustrates the problem when using a continuous planner for ground navigation; a terrain map is given where each cell has an associated cost, or traversibility difficulty. The AGV seeks to find a path to its goal cell by only considering a limited horizon. The pink arrow represents a path that the AGV would take if it does not consider the entire course; a dead end.

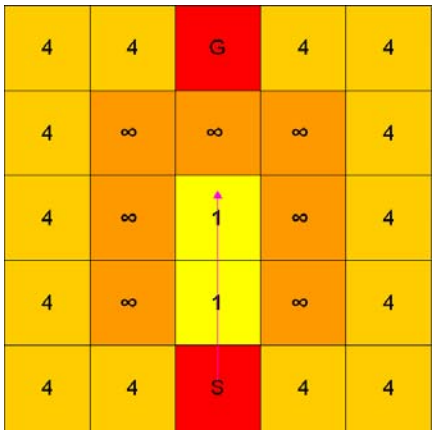

Figure 4: Flaw of Continuous Process Implementation on AGV

Graph Model Concepts

Before any detailed discussion on path theory can begin in earnest, it is important to outline the standard notation used to explain concepts in this largely mathematical field. The following pages define a number of terms, and establish the notation that will be used throughout this paper.

Terminology

A *graph G* is defined to be a set of nodes, denoted by *S*, and a set of directional edges, denoted by *E*. If ε_{ij} is an element of $\boldsymbol{E}$, it is said that there is an *edge* from node $s_i \in S$ to node $s_j \in S$, s_j is a *successor* of s_i, and s_i is a *predecessor* of s_j. Note: the existence of ε_{ij} does not imply the existence of ε_{ji}. The graphs of concern for this application are those whose edges have *costs* associated with them. The cost of edge ε_{ij} is denoted by $c(s_i, s_j)$. Note: the existence of ε_{ij} and ε_{ji} does not imply that $c(s_i, s_j) = c(s_j, s_i)$. In this report, the only graphs of concern are those where for every pair of nodes s_p and s_q that are elements of *S*, if the edge ε_{pq} exists, the cost of ε_{pq} is greater than zero. That is, $\forall\ s_i, s_j \in S$, $\varepsilon_{ij} \neq \varnothing \rightarrow c(s_i, s_j) > 0$.

Define the successor multi-valued function Γ on S, whose value for each s is the set of successors $S' \subset S$ of s. That is, $\Gamma(s) = S' : \forall\, s' \in S', \boldsymbol{\varepsilon}_{s,s'} \neq \varnothing \ \wedge \forall\, s'' \in \{x \in S : x \notin S'\}$, $\boldsymbol{\varepsilon}_{s,s''} = \varnothing$. Let the inverse of Γ, Γ^{-1}, when applied to s, yield the set of predecessors of s. When Γ or Γ^{-1} is applied to a node s, it is said that s is *expanded.*
A *path* $p(s_1, s_n)$ from s_1 to s_n is an ordered set of nodes $\{s_1,s_2,\ldots,s_i,s_{i+1},\ldots,s_n\}$ with each s_{i+1} a successor of s_i. The cost of a path is the sum of the cost of the edges in the path. That is, $c(p(s_1, s_n)) = c(s_1,s_2) + c(s_2,s_3) + \ldots + c(s_i,s_{i+1}) + \ldots + c(s_{n-1},s_n)$. If a path from s_i to s_j exists, s_j is said to be *accessible* from s_i. The problem of finding an optimal path from a starting node s_{start} to a goal node s_{goal} can be stated as such:
If s_{goal} is accessible from s_{start}, find the cheapest path $p(s_{start}, s_{goal})$. That is, given $s_{start},s_{goal} \in S$, find $\text{argmin}(c(p(s_{start}, s_{goal})))$.

Finding an Optimal Path

A classical approach to finding the optimal path in a graph is to use Dijkstra's algorithm (Algorithm). Define the path cost function $g : S \to \mathbb{R}^+$ to yield the cost of the cheapest path from the start node s_{start} to the present node. That is, $g(s) = \min\{c(p(s_{start}, s))\}$. $g(s)$ is also referred to as the *g-value*. Dijkstra's algorithm initializes *g*-values for every node, and then improves them with each iteration. To access the optimal path after the process is complete, use Algorithm 2.

Algorithm 1: Dijkstra's Algorithm	
Step 1	Assign the start node a g-value of zero and every other node a g-value of infinity.
Step 2	Mark all nodes *unvisited* and set the initial node to the current node, s.
Step 3	Expand s and update the g-values for every successor node s' that is *unvisited*. That is, $g(s') = g(s) + c(s, s')$.
Step 4	Mark s *visited* and set the unvisited node in the graph with the smallest g-value as the current node s.
Step 5	Repeat steps 3 and 4 until $s = s_{goal}$.

Algorithm 2: Constructing Optimal Path	
Step 1	Set the current node s' to s_{goal}.
Step 2	Expand s'.
Step 3	Connect s' to the predecessor node s with the smallest g-value, and set s' to s.
Step 4	Repeat steps 2 and 3 until $s' = s_{start}$.

Dijkstra's algorithm always yields the optimal path from a start node to a goal node. However, it is inefficient because many nodes are expanded, which makes for long computation time. Consider a situation where information is known about the problem's environment such that even without expanding a node s, an approximate cost can be determined for the optimal path from s to s_{goal}. This approximate cost is called the *heuristic*, or *h-value*. The sum of the *g*-value and *h*-value is called the *f-value*. That is, $f(s) = g(s) + h(s)$. The *f*-value of a node can be thought of as the lower bound of the cost of the path from s_{start} to s_{goal} that goes through s. Then, instead of visiting every node, the only nodes that would need to be visited are ones that have the smallest *f*-value.

To illustrate an example where using a heuristic would greatly reduce the number of nodes expanded in a path search, consider the problem of finding the fastest road route from an initial city s_{start} to goal city s_{goal}. Using a graph to model the problem would be appropriate, where edges represent streets and nodes represent intersections. The cost of each edge would be the travel time of traversing it. Because it is impossible for a road to be shorter than the perfectly straight line connecting a node s to s_{goal}, an appropriate h-value would be the travel time of the straight-line path from a node s to s_{goal}, which is the lower bound of the travel time between the two points. To see how this heuristic would be useful, consider the following case:

Suppose the current node s is the start node s_{start}, and is expanded, and there are two successors, s_1 and s_2, where $g(s_1) < g(s_2)$. Using Dijkstra's algorithm's would always be set to s_1. Suppose that s_{goal} is directly north of s_{start}, there is a straight road connecting them, s_2 is an intersection on that road, and s_1 is an intersection that is exactly east of s_{start}, but is closer than s_2 (see Figure 5).

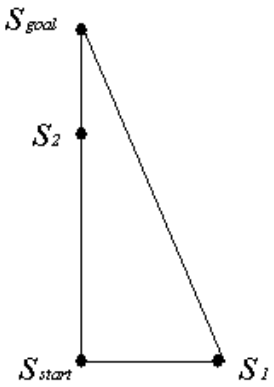

Figure 5: Case where g(s1) < g(s2) but s2 is a Node on the Optimal Path.

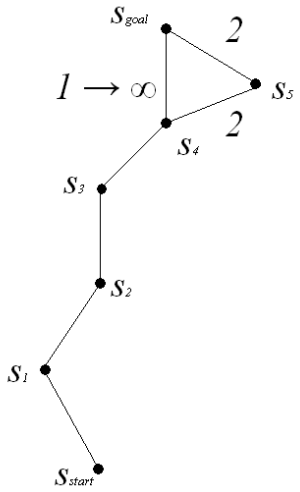

Figure 6: Case Where Change in Environment Does Not Drastically Affect the Optimal Path

Assuming the road conditions are homogenous (traffic, speed limit, etc. is the same everywhere), it can be seen that without taking the h-value into account, the non-optimal node, s_1 would be expanded next, but if f-values of unvisited nodes were compared instead of g-values, s would be changed from s_{start} to s_2, which would be the optimal choice, and would thereby not waste time on an unnecessary node expansion. Algorithm 3 is a heuristic extension of Dijkstra's algorithm known as A*.

Algorithm 3: A*	
Step 1	Assign the start node an *f*-value of $h(s_{start})$ and every other node an *f*-value of infinity.
Step 2	Mark all nodes *unvisited* and set the initial node to the current node, *s*.
Step 3	Expand *s* and update the *f*-values for every successor node *s'* that is *unvisited*. That is, $f(s') = g(s) + c(s, s') + h(s')$.
Step 4	Mark *s visited* and set the unvisited node in the graph with the smallest *f*-value as the current node *s*.
Step 5	Repeat steps 3 and 4 until $s = s_{goal}$.

In cases where the environment is not static, the optimal path needs to constantly be modified. When most of the changes do not drastically affect the entire optimal path, the path does not necessarily have to be reconstructed from scratch. For example, consider the case in Figure 6 where initially, the optimal path from s_{start} to s_{goal} takes the road modeled by the edge from s_4 to s_{goal}. Suppose, however, that while the vehicle is still very far from s_{goal}, there is an accident on the said road. To update the optimal path, a simple detour would need to be made and instead of traveling straight to s_{goal} from s_4, the optimal path would travel from s_4 to s_5, and then to s_{goal}. But, the optimal path from s_{start} to s_4 would remain the same.

While an AV navigates, the most drastic changes in the environment are typically close to it. This is because the global map that is provided via some agent has error, and the perception of the local environment picked up by sensors on the AV give a much better picture of the local environment. Therefore, path-replanning algorithms typically find an optimal path starting from the goal and work towards the present location, so that local changes do not affect most of the optimal path. In this case, the *g*-value refers to the cost of the optimal path from a node *s* to s_{goal} and the *h*-value refers to the Euclidean distance from s_{start} *to s*. Using A* to find an initial optimal path but starting with s_{goal} and working towards s_{start} is called *Backwards* A*.

One dynamic re-planning extension of A* is called D* Lite, which introduces additional terminology. When the cost of an edge changes, the *g*-value of the successor usually changes, and the node whose *g*-value changes is said to be *inconsistent*. If $c(s, s')$ changes and $g(s')$ decreases, *s'* is said to be *over-consistent*. On the other hand, if $c(s, s')$ changes, $g(s')$ increases and *s'* is said to be *under-consistent*. Algorithm 4 shows a high level description of D* Lite (note: this is a version not optimized for computation time; for an optimized version see Koenig & Likhachev). To access the optimal path after D* Lite is complete, use Algorithm 2 but instead of starting at s_{goal}, start at s_{start} and work towards s_{goal}.

Algorithm 4: D* Lite	
Step 1	Use Backwards A* to find an optimal path from s_{goal} to s_{start}.
Step 2	Update the *g*-value for every node *s'* where *c*(*s*, *s'*) changed for a predecessor node *s* by considering each predecessor node *p* (going towards s_{goal} from *s'*) and finding the one that minimizes *g*(*p*) + *c*(*p*, *s'*), which equals the new *g*(*s'*) (note: *p* might equal *s*).
Step 3	While the inconsistent node *s* with the smallest *f*-value is less than the *f*-value of s_{start}: • If *s* is under-consistent, repeat Step 2 for every predecessor of *s* (towards s_{start}). • If *s* is over-consistent, repeat Step 2 for *s* and every predecessor of *s*.

It should be noted that D* Lite does not necessarily derive an optimal path faster than A*. This is because A* expands each node at most, once, while D* Lite can expand a node twice: once as an under-consistent state and once as an over-consistent state. Thus, D* Lite should only be used if the environment is mostly static, and only minor changes need to be made to correct the optimal path.

Approach to Amphibious Navigation

The only way to develop, or identify a potential process for amphibious navigation is to separate the problem field into its three components; sea, land and transition. By investigating what options exist in planning paths in each, while considering the requirements of interactivity, it may be possible to identify a hybrid algorithm, or set of algorithms which can be used to provide full amphibious autonomy.
To this end, the problem is considered as separate tasks; having the start and end points both at sea, both on land, and only then starting at sea and terminating on land (and vice versa).

Sea Surface Navigation

An AuAV, like an ASSV, will be put through extensive trials to identify its operational capacity – generally conducted on prototypes before the final design is built. The analysis of this prototype data should be used to derive an estimated direction dependent speed function that takes information from the vehicle's sensor suite (such as roll angle, wind velocity, and its own velocity) as inputs.

The direction dependent speed function $V(\theta): [0, 2\pi] \rightarrow \mathbb{R}^+$ denotes the maximum attainable speed for a given heading angle θ. Let P_{xy} be defined as the set of all continuous and rectifiable paths from the start point x to a target point y. Define the path traversal time function $t: P \rightarrow \mathbb{R}^+$, where $\forall\, p \in P_{xy}$, $t(p)$ denotes the travel time required to traverse the path p. The problem can now formally be defined:
For a given speed function $V(\theta): [0, 2\pi] \rightarrow \mathbb{R}^+$, a starting point $x \in \mathbb{R}^2$, and a goal point $y \in \mathbb{R}^2$, find a fastest path from x to y that lies in $\mathbb{R}^2$. That is, find $p' \in P_{xy} : t(p') \leq t(p)\ \forall\ p \in P_{xy}$.

An AuAV must operate in shallow water and the surf zone, where predicting the dynamics requires very sophisticated modeling techniques (Madsen *et al.*). Since autonomous sea surface navigation requires continuous re-planning, as opposed to batch planning, as discussed earlier, incorporating the models into a navigation algorithm would be infeasible. To develop an intuitive, computationally feasible algorithm, the problem can be simplified by treating the sea state in the ocean as time and space homogeneous every time a direction dependent speed function is calculated. In other words, whenever $V(\theta)$ is calculated at the vehicles present location, it is assumed that every point in the sea area of operation has the same direction-dependent speed function.

The homogenous simplification allows for the consideration of a Unit Linear Path Attainable Region (ULPAR), or the set of all points that can be reached in a unit time period from a starting point while following a straight line path. That is, $\forall\, x \in \mathbb{R}^2$, an ULPAR $L(x) := \{y \in \mathbb{R}^2 : \| y - x \| \leq V(\theta_{y-x})\}$, where θ_{y-x} and $\| y - x \|$ denote the angle and length of a vector $y - x$, respectively. In the case where the ULPAR is convex, the optimal path in P_{xy} is merely the straight line connecting points x and y. However, in most cases, the ULPAR will not be convex, so a theorem developed by Dolinskaya & Smith (2008a) is applied (see Theorem 1). Some additional terminology is required:

Define $bd(L(x))$ to be the border of $L(x)$ and $conv(L(x))$ to be the convex hull of $L(x)$. For the problem of finding the fastest path from x to y, let k be the point of intersection of the line l_{xy} connecting points x and y, and the border of the ULPAR $L(x)$, i.e., $k := l_{xy} \cap bd(L(x))$. Similarly, $k' := l_{xy} \cap bd(conv(L(x)))$. See Figure 7 and Figure 8 for a diagram illustrating these definitions.

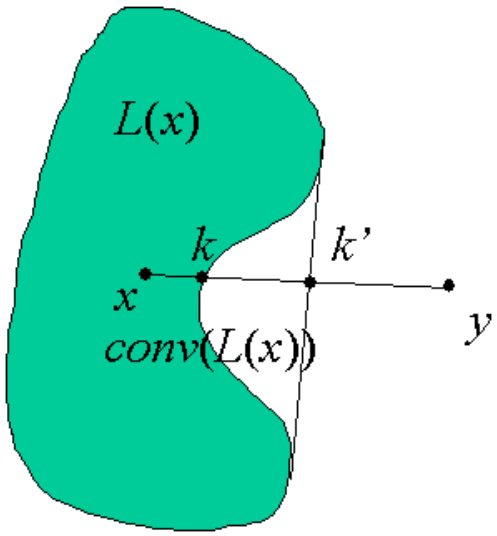

Figure 7: ULPAR Definitions Illustration

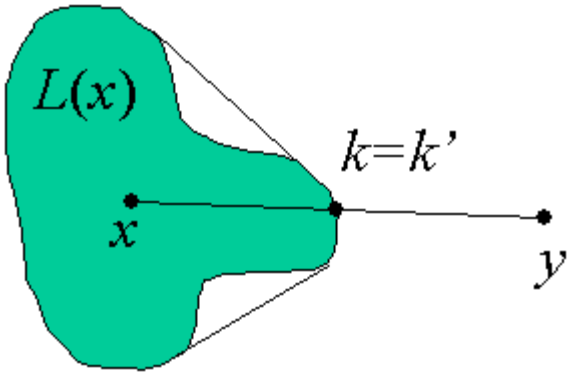

Figure 8: Case 1, k = k'

Theorem 1: Optimal Path between Two Points for Non-Convex ULPAR (Dolinskaya & Smith, 2008a)
If $k = k'$, the fastest path from x to y is the straight line segment connecting them (see Figure 8). If $k \neq k'$, the fastest path from x to y consists of two line segments: the straight line segment from point x to point $z = x + \alpha\lambda(x_1 - x)$ and the second line segment from point z to point y, where $\alpha = \frac{\|y-x\|}{\|k'-x\|}$ and $x_1, x_2 \in L(x)$ s.t. $\exists\ \lambda \in [0,1] : k' = \lambda x_1 + (1 - \lambda)x_2$ (see Figure 9, and note that $(y - z) \| (x_2 - x)$).

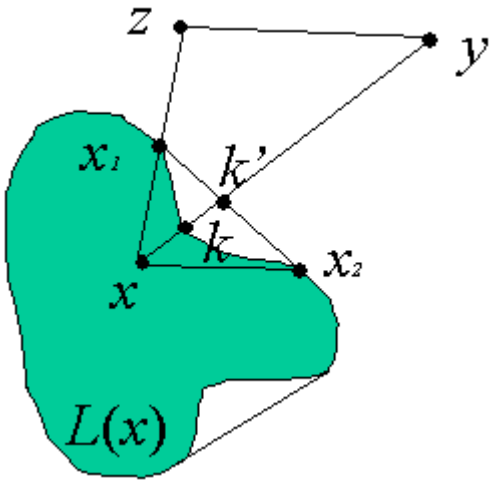

Figure 9: Case 2, k ≠ k'

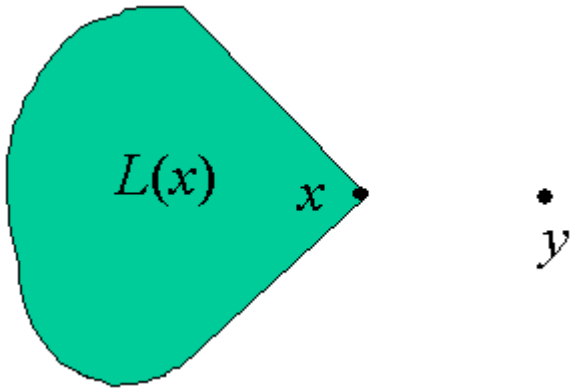

Figure 10: Example of Case with No Feasible Path

The fastest path theorem stated above assumes that a feasible path from x to y exists. In some situations, the sea state might be such that no feasible path exists with the corresponding ULPAR (see Figure 10). In the event that DUKW-ling has no feasible path from its present location to its goal, it must go backwards until a feasible path exists. The planner's first step will always be to see if a feasible path exists by using the approach laid out in the following:

If the domain of $V(\theta)$ is extended to $[-\pi, 3\pi]$, lower and upper heading angle bounds can be defined. If $V(\theta_{y-x}) = 0$, define the lower heading angle bound θ_L to be the greatest angle going clockwise on V starting at θ_{y-x} where the maximum attainable speed at every angle between this and θ_{y-x} is zero. That is, $\theta_L := \inf\{\theta^* : V(\theta) = 0, \forall\ \theta \in [\theta^*, \theta_{y-x}]\}$. Similarly, the upper heading angle bound $\theta_U := \sup\{\theta^* : V(\theta) = 0, \forall\ \theta \in [\theta_{y-x}, \theta^*]\}$. See Figure 11 for an illustration of the definitions. To determine if a feasible path exists from x to y, Theorem 2 is used.

Theorem 2: Path Feasibility (Dolinskaya & Smith, 2008a)
A feasible path from x to y does not exit if and only if $V(\theta_{y-x}) = 0$ and $\|\theta_U\| + \|\theta_L\| \geq \pi$.

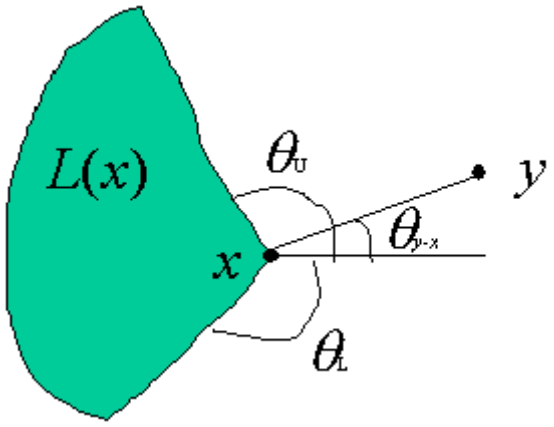

Figure 11: Illustration of Defined Heading Bounds

Now consider the problem of finding the fastest path from point s to point t in a plane with polygonal domain, $\mathbb{R}^2 \setminus P$, where P is the set of obstacles, and $L(s)$ is convex. Let $\tau_V(i,j)$ be the travel time from point i to point j for a direction dependent speed function V. Algorithm 5 finds an obstacle avoiding fastest path when the ULPAR is convex (modified from Dolinskaya & Smith, 2008b). Backwards A* was chosen as the algorithm for finding the optimal path in the visibility graph instead of D* Lite because

the cost of almost every edge in the graph will change as *V* changes, which would make D* Lite slower due to the double node expansion mentioned earlier.

Algorithm 5: Obstacle-Avoiding Fastest Path for Convex ULPAR	
Step 1	Construct a visibility graph *G* as follows (see Figure 12 for an example): • The set of nodes, *S*, is composed of all the vertices of the obstacles in *P*, as well as start point *s* and target *t*. • The set of edges, *E*, consists of all the straight-line edges interconnecting these vertices such that they do not intersect any of the obstacles in *P*. • For an edge with parent node *i* and successor node *j*, let the cost of the edge, $c(i,j) = \tau_V(i,j)$.
Step 2	Apply Backwards A* Algorithm to find an optimal path in *G* from node *s* to node *t*. For the *h*-value, use the travel time of traveling the Euclidean distance from *s* to the present location (this the lower bound of the fastest route). The resulting path is an obstacle-avoiding fastest path. See Figure 13 for an example of an obstacle avoiding fastest path for the visibility graph in Figure 12.

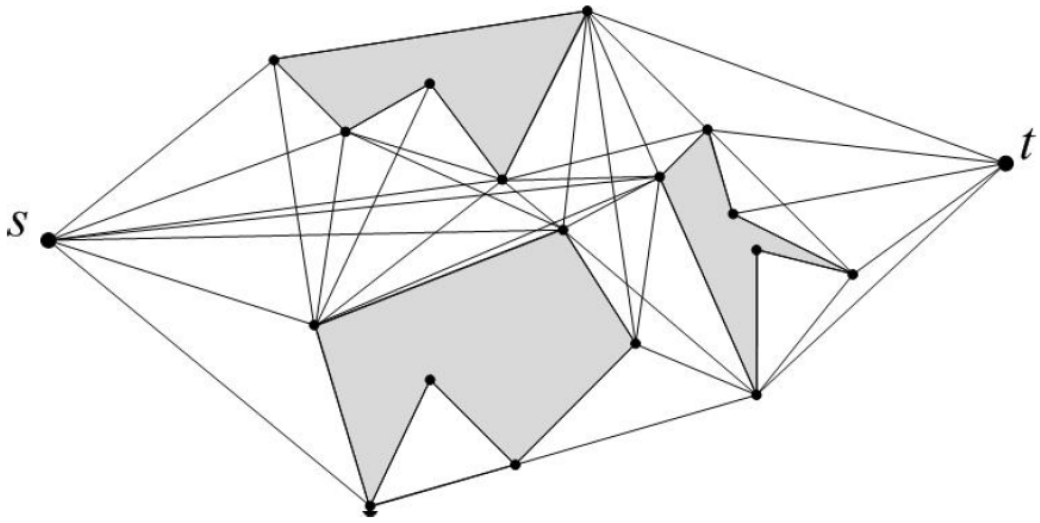

Figure 12: Construction of a Visibility Graph (Dolinskaya & Smith, 2008b)

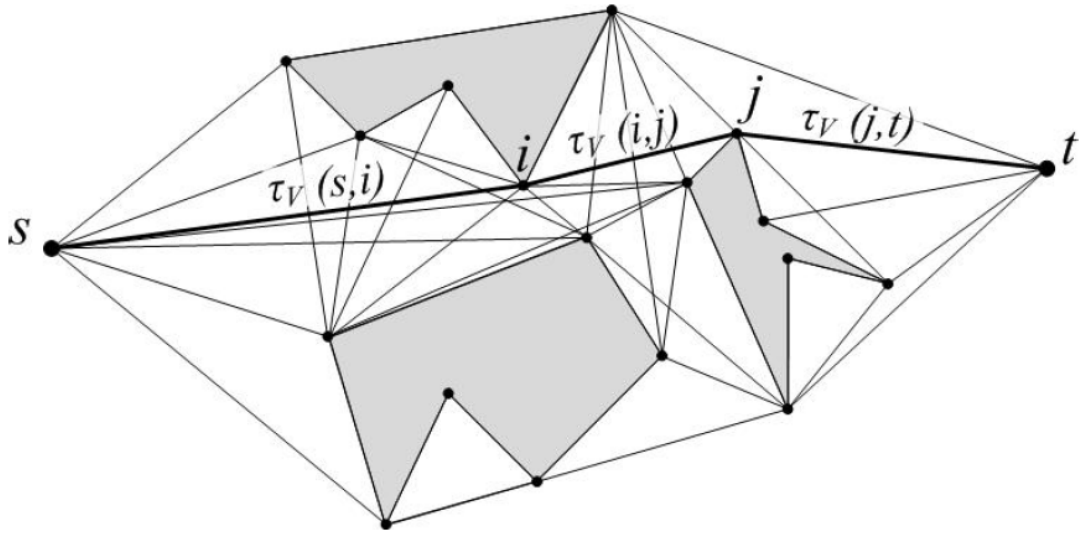

Figure 13: Optimal Path for Visibility Graph in Figure 12 when L(s) is Convex (Dolinskaya & Smith, 2008b)

To extend this approach to the case when $L(s)$ is not convex, apply Algorithm 5 using the convex hull of V, denoted as V'. Then use Theorem 1 to find an optimal path from the start point to the end point of each edge in the path obtained from Algorithm 5. Denote this transformation with the function $\omega : E \rightarrow E \text{ x } E$, where $\omega(\varepsilon_{xy}) = \{\varepsilon_{xz}, \varepsilon_{zy}\}$, where $\tau_{V'}(x, y) = \tau_V(x, z) + \tau_V(z, y)$ (for a proof, see Dolinskaya & Smith, 2008a).

However, consider the case where the optimal path corresponding to an edge is blocked by an obstacle. To deal with this problem, divide the edge into subsections such that the optimal path corresponding to each subsection is not blocked by an obstacle. That is, $\forall \varepsilon_{xy} \in E, \omega(\varepsilon_{xy}) \cap P \neq \varnothing \rightarrow \exists n \in \mathbb{N}$ *s.t.* $\tau_{V'}(x, p_1) + \tau_{V'}(p_1, p_2) + \ldots + \tau_{V'}(p_n, y) = \tau_{V'}(x, y) \ \wedge \ \omega(\varepsilon_{x,p1}) \cap P = \varnothing$. The optimal path is then the sum of these zigzag paths. See Figure 14 for an illustration of such a path.

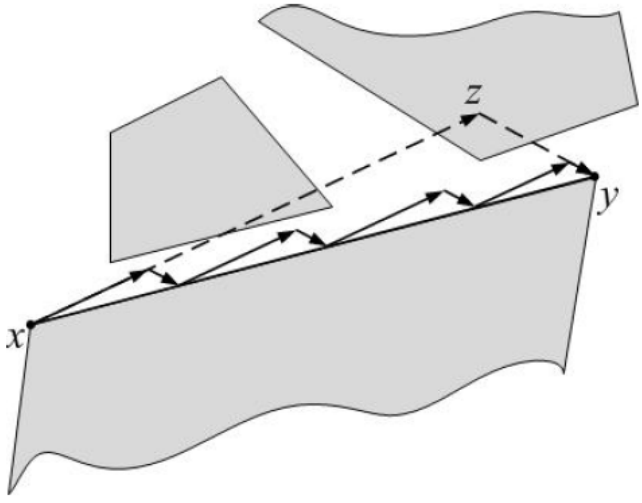

Figure 14: Optimal Path from x to y (Dolinskaya & Smith, 2008b)

Using the mentioned theorems and processes, Dolinskaya & Smith derive an algorithm which finds an optimal path from *s* to *t* such as in Figure 15. But, since the optimal path still intersects the nodes of the path yielded by Algorithm 5, to save computing time, it will only be of interest to derive the optimal path from the present location to the nearest node in the path from Algorithm 5. If the travel time of the path to the first node is less than the time it takes the computer to find a new path, the path should be calculated through the second node. Now Algorithm is introduced, which finds the heading for the optimal path from a start point in water to a goal point in water.

Algorithm 6: Heading for Optimal Path	
Step 1	Find $V'(\theta)$ for $\theta \in [0, 2\pi]$ such that $L_{V'} = conv(L_V)$.
Step 2	Use Algorithm 5 to find an optimal path $p_{V'}$ corresponding to V' .
Step 3	Let $\{k_0, k_1, k_2, \ldots, k_n\}$ be an ordered set of vertices $p_{V'}$ traverses where $k_0 = s$ and $k_n = t$. For the pair k_0, k_1, find $\omega(\varepsilon_{k0,k1})$. If $\omega(\varepsilon_{k0,k1}) \cap P \neq \varnothing$, divide $\varepsilon_{k0,k1}$ into m subsections such that $\tau_{V'}(k_0, p_1) + \tau_{V'}(p_1, p_2) + \ldots + \tau_{V'}(p_m, k_1) = \tau_{V'}(k_0, k_1)$ and $\omega(\varepsilon_{k0,p1}) \cap P = \varnothing$. Use this approach to find the zigzag optimal path and make this the new $\omega(\varepsilon_{k0,k1})$.
Step 4	Navigate along $\omega(\varepsilon_{k0,k1})$. If the optimal $\tau_V(k_0, k_1)$ is less than the time it takes to plan a new path, repeat step 3 to find $\omega(\varepsilon_{k1,k2})$ and add that path to $\omega(\varepsilon_{k0,k1})$.
Step 5	In parallel, process the following, and navigate on the path that is derived first: • Repeat Steps 3-4 to find $\omega(\varepsilon_{k0,k1})$ and/or $\omega(\varepsilon_{k0,k2})$. • If Step 2 is not being processed, start processing Step 2.

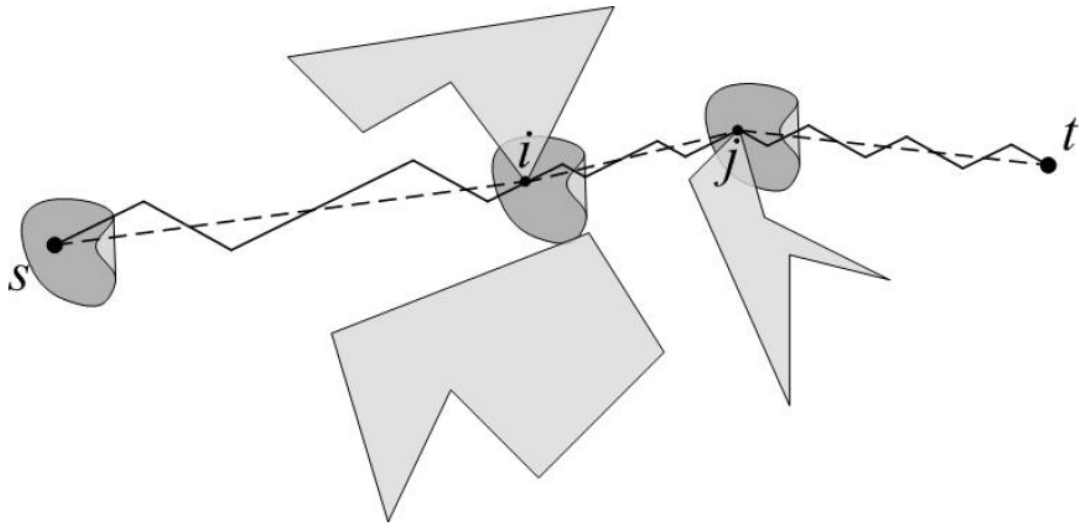

Figure 15: Optimal Path for Visibility Graph in Figure 12 when L(s) is Non-Convex (Dolinskaya & Smith, 2008b)

Ground Navigation

An AuAV, like typical military AGVs, will have a grid-based representation of a map containing information about the terrain, as well as enemy locations. Each cell in the grid will have an associated cost corresponding to its traversibility difficulty (recall Figure 4). For ground navigation, the task is to plan the path of least net cost from the AV's present location in the grid to a desired goal location in the grid.

A common method of approximating the grid is with a graph, where nodes represent grid cell centers and edges connect nodes corresponding to adjacent grid cells. This simplifies the problem of path planning to finding the optimal path of a graph which was discussed earlier (see Ferguson *et al.* for a survey). A significant limitation of this approach is that the paths produced are restricted to headings of $\pi/4$ increments. See Figure 16 and Figure 17 for an illustration of this problem reproduced from Ferguson & Stentz, 2005. In Figure 17, shaded cells indicate obstacles, the black line is the angle-limited path, and the dashed blue line is the optimal path.

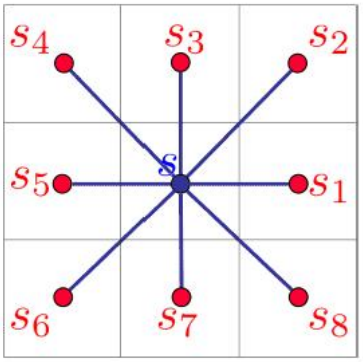

Figure 16: A Grid in which Nodes Reside at the Center of each Cell (Ferguson & Stentz, 2005)

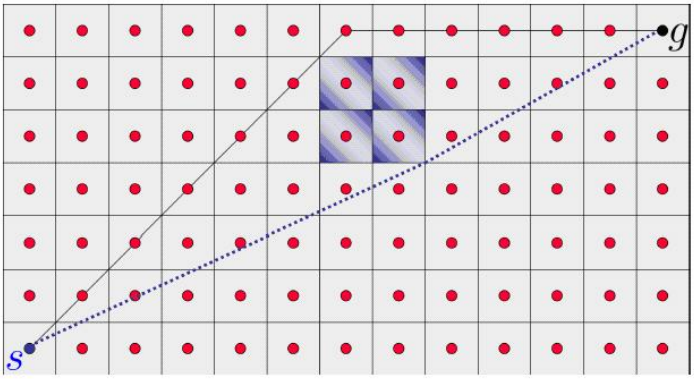

Figure 17: Comparison of Paths (Ferguson & Stentz, 2005)

A state-of-the-art path planner and re-planner, Field D* (Ferguson & Stentz, 2005) is introduced in the following discussion, which assigns nodes to represent cell corners instead of its center (see Figure 18). Field D* has been successfully implemented on many AGVs, including Mars Exploration Rovers (Carsten *et al.*), and it is the algorithm that an AuAV should use for ground navigation. Additional terminology is required to understand Field D*.

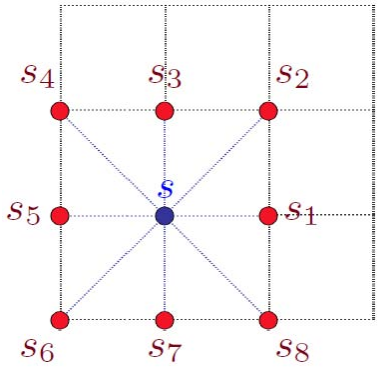

Figure 18: A Grid in which Nodes Reside at Cell Corners (Ferguson & Stentz, 2005)

Terminology

A cell κ is an integer point that contains a positive traversal cost c, and a set of four corner nodes, denoted by S_c. That is, $\kappa := \{\mathbb{Z}^2, \mathbb{R}^+, \{S_c\}\}$. A set of cells will be referred to as a *grid*. To formally state the problem of finding an optimal ground path:
Given a grid and two nodes s_{start} and s_{goal} in the grid, find the path within the grid from s_{start} to s_{goal} with minimum cost.

Define the node neighbor pair finding function $\mu : S \rightarrow \{(S, S)\}$ that would return a set of *node neighbor pairs* (NNP). For example, in Figure 18, $\mu(s)$ would return the set $\{(s_1,s_2),(s_2,s_3),\ldots,(s_7,s_8),(s_8,s_1)\}$, where each (s_i, s_j) is a NNP.

Calculating Path Cost

With algorithms where nodes correspond to grid cell centers, the cost of traveling from one node to the successor node in a neighboring cell is the cell cost corresponding to the successor node. This allows for the straightforward g-value from D*Lite and Backwards A*: $g(s) = \min\{c(s, s') + g(s') \; \forall \; s' \in \Gamma(s)\}$.

With Field D*, calculating the traversal cost from one node to another is less intuitive which makes finding g-value more difficult. Field D* uses linear interpolation to arrive at the following path cost function for $g(s_y)$, where s_y is a point on the cell edge with endpoints s_1 and s_2 (see Figure 19): $g(s_y) = yg(s_2) + (1 - y)g(s_1)$, where y is the distance from s_1 to s_y, assuming unit cells. To reiterate, $g(s_i)$ is the cost of the cheapest path from s_i to s_{goal}. Staying with the example in Figure 19, to find $g(s)$, the cost of traveling from s to s_y must be added to $g(s_y)$.

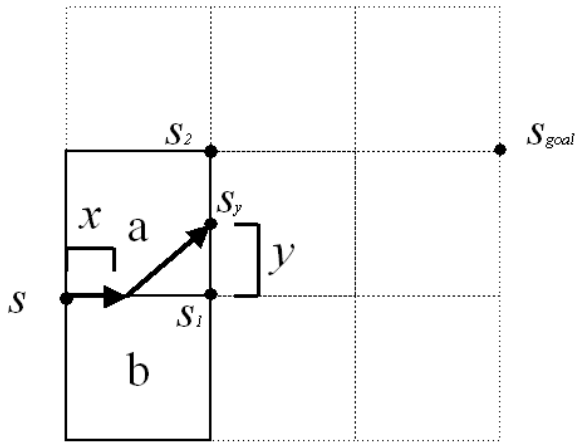

Figure 19: Illustration of Path Cost Function

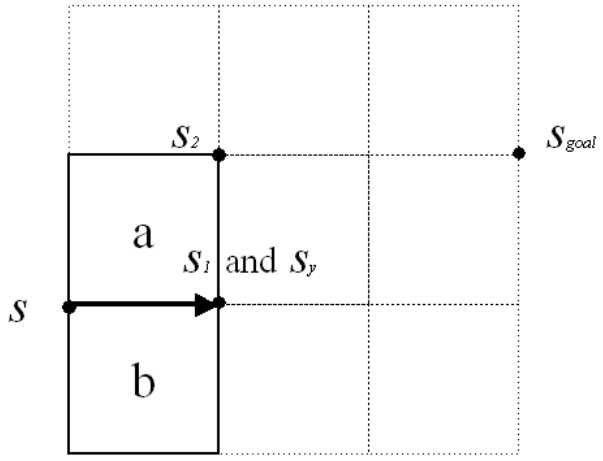

Figure 20: Optimal Path from s to sy if $\delta < 0$

The problem of finding the cost of traveling from s to s_y is generalized to determining the cheapest cost for a path from s to any point on the line segment $l_{1,2}$ connecting the points at s_1 and s_2. The path cost is a function of two cell weights, as well as the distance in the x and y direction. Consider the example path in Figure 19. The path cost $c(s, l_{1,2}) = bx + a\sqrt{(1-x)^2 + y^2}$, where a and b are weights of their corresponding cells.

It has been shown that there are only three possible optimal path forms from s to s_y (see Figure 20, Figure 21, Figure 22) (Ferguson & Stentz, 2005). Let δ be the difference in node g-values. That is, $\delta(s_1,s_2) = g(s_1) - g(s_2)$. Algorithm is the process of calculating the path cost for a node s going through a line segment $l_{1,2}$. Calculating the path cost requires three node parameters. $c(s, s_1, s_2)$ yields the cost of the path from s to the cheapest point on $l_{1,2}$ where s_2 is diagonal to s, a is the traversal cost of the cell with corners s, s_1, s_2, and b is the traversal cost of the cell with corners s, s_1, but not s_2.

Algorithm 6: Calculating $c(s, s_1, s_2)$	
Step 1	Check if a and b are both infinity. $a \wedge b = \infty \rightarrow g(s) = \infty$.
Step 2	Find $\delta = g(s_1) - g(s_2)$. Depending on the value of δ, $g(s)$ will have one of the following values: • If $\delta < 0$, then the optimal path from s through $l_{1,2}$ travels straight through s_1 (Figure 20), and has a path cost $g(s) = \min(a, b) + g(s_1)$. • If $0 < \delta \leq b$, then the optimal path travels straight through a point on $l_{1,2}$ not s_1 or s_2 (Figure 21), and has a path cost $g(s) = \sqrt{1+y^2} + \delta(1-y) + g(s_2)$, where $y = \min\left(\frac{\delta}{\sqrt{a^2-\delta^2}}, 1\right)$. • If $0 < \delta \leq b$, then the optimal path travels some of the bottom edge and then straight through s_2 (Figure 22), and has a path cost

$$g(s) = a\sqrt{1+(1-x)^2} + bx + g(s_2)\text{, where}$$
$$x = 1 - \min\left(\frac{b}{\sqrt{a^2-b^2}}, 1\right).$$

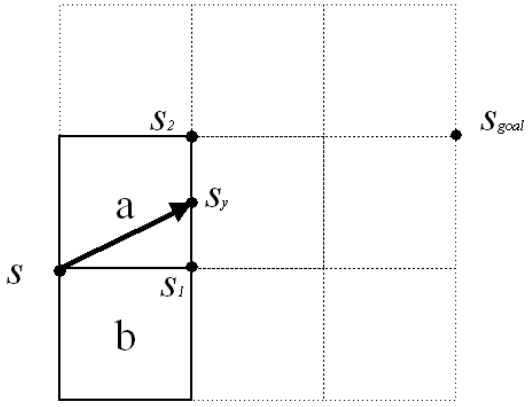

Figure 21: Optimal Path from s to sy if $0 < \delta \leq$ b

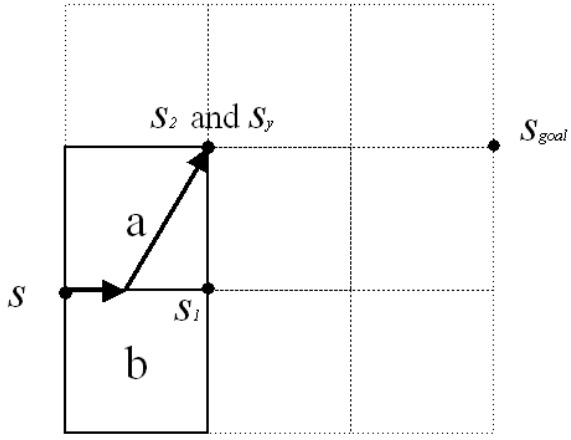

Figure 22: Optimal Path from s to sy if b < δ

Deriving the Path

Field D* works similarly to D* Lite, but since it is not a graph path planner, there are some differences. First, in deriving the initial path, Backwards A* cannot be used so it is extended to work in the cell node model used by Field D* (see Algorithm 7). Second, the heuristic must be modified to take into account that a point on a path can be changed to any point along that cell side. Thus, the *h*-value of a node *s* is the minimum Euclidean distance from s_{start} to any point on the side of the cell containing *s*. To trace the optimal path, use Algorithm , which is an extended version of backwards Algorithm 2 to the grid problem domain. Algorithm is a high level description of Field D* (note: this is an un-optimized version; for an optimized version see Ferguson & Stentz, 2005).

Algorithm 7: Backwards A* Extended to Grid Problem Domain	
Step 1	Assign the goal node an *f*-value of $h(s_{goal})$ and every other node an *f*-value of infinity.
Step 2	Mark all nodes *unvisited* and set the initial node to the current node, *s*.
Step 3	Expand *s* and update the *f*-values for every successor node *s'* that is *unvisited*. That is, *f*(*s'*) = *g*(*s*) + *c*(*s*, *s'*,*s''*) + *h*(*s'*) (use Algorithm to find *c*(*s*, *s'*,*s''*)).
Step 4	Mark *s visited* and set the unvisited node in the graph with the smallest *f*-value as the current node *s*.
Step 5	Repeat steps 3 and 4 until $s = s_{goal}$.

Algorithm 8: Constructing Optimal Path in a Grid	
Step 1	Set the current node s to s_{start}.
Step 2	Expand s.
Step 3	Connect s to the point which minimizes $g(s) + c(s,s',s'')$ for all NNP (s',s''). Set s to s'.
Step 4	Repeat steps 2 and 3 until $s = s_{start}$.

Algorithm 9: Field D*	
Step 1	Use Backwards A* to find an optimal path from s_{goal} to s_{start}.
Step 2	Update the g-value for every node s' where $c(s, s',s'')$ changed for a node s by considering each node p (going towards s_{goal} from s') and finding the one that minimizes $g(p) + c(p, s',s'')$, which equals the new $g(s')$ (note: p might equal s).
Step 3	While the inconsistent node s with the smallest f-value is less than the f-value of s_{start}: • If s is under-consistent, repeat Step 2 for every s' (towards s_{start}). • If s is over-consistent, repeat Step 2 for s and every s'.

Transition

While the ground-to-ground path planning algorithm is a standard AGV algorithm, and the sea-to-sea algorithm is derived from sea path optimization theorems, there has been little to no research on planning paths for the transition from sea to ground and vice versa.

The major challenge in transitioning is identifying the best beaching point. Recall that finding an optimal sea path requires finding a simplified path from the present location to a target point. Consider the case illustrated in Figure 23, where the AuAV must decide which point on shore to beach. The left path at first may appear to be optimal because it is in the heading angle of least resistance (as can be seen in the ULPAR), and the beaching point has the lowest traversibility cost. However, the path would require the vehicle to travel a considerable distance in navigating around the wall of obstacles once it has beached. If it took the path on the right, it would incur a greater cost at first, but would have an optimal path to travel to the final target from the beaching point. The proposed algorithm solves this problem differently for each type of transition (sea-to-ground and ground-to-sea).

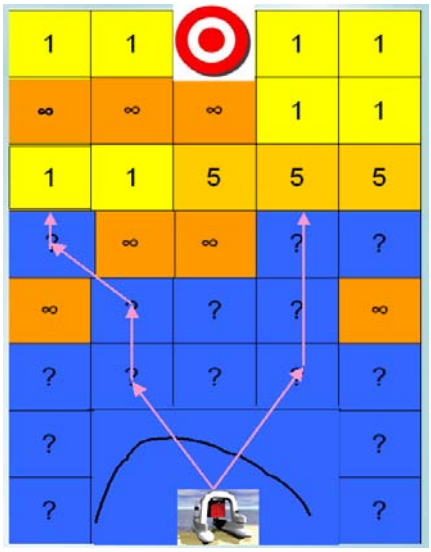

Figure 23: Illustration of Difficulty in Identifying Beaching Point

Sea-Surface to Ground

The first approach considered for sea-surface to ground navigation is to find an optimal path that considers both sea and land conditions. Since the optimal sea path is based on a visibility graph with weighted edges, an intuitive way of incorporating information from the ground is to integrate it with a ground path planner that uses a graph model of the grid, such as D* Lite. Algorithm is a possible algorithm that integrates the two graph models.

Algorithm 10: Optimized Sea-Surface to Ground Path Plan	
Step 1	Construct a graph representation of the grid model of the ground using cell centers as nodes and traversibility cost as edge cost.
Step 2	Use Algorithm 5 to determine the optimal travel time from the vehicle's present location to each node on the shore. Construct an edge from the vehicle's location to each node on the shore where the edge cost is a function of the optimal travel time to that node.
Step 3	Use D* Lite (Algorithm 4) to find the optimal path from the present location to the goal point on ground.
Step 4	Use Algorithm to find the heading of the optimal path from the present location to the point on shore that is on the optimal path found in Step 3.
Step 5	While the vehicle is in the water and surf zone, repeat steps 2-4.
Step 6	When the vehicle is firmly on ground, use Field D* (Algorithm) to find the optimal path to the goal.

Algorithm would come close to finding an optimal path that takes into account both sea and ground conditions, however it would be computationally exhaustive because of Step 2, where separate visibility graphs are constructed for each shore point that is used as a terminating waypoint. To derive a truly optimal path, the ground would need to be represented by a high resolution grid, which would increase the amount of graphs that would need to be constructed and then searched with D* Lite. An option would be to

create a low resolution grid approximation of the ground, which would result in less graphs being constructed, but then the path would not be very optimal and would still take more time than if only a single graph were constructed.

Algorithm 11 shows a less optimal, though much more computationally efficient approach to finding an effective path from a start point at sea to a goal point inland. It ignores ground traversibility difficulty when $c < \infty$ while the vehicle is still in water (see Figure 24). This is done because it is believed that since surf zone dynamics are so chaotic, any extra time it would take to optimize an amphibious path that takes into account ground conditions would be better spent coming up with a new plan that would adjust the vehicles heading to better travel in a new sea state. Furthermore, with Algorithm 11, when the vehicle beaches, a feasible path will exist to the goal waypoint, and Field D* would optimize the remainder of the path.

Algorithm 11: Computationally Feasible Sea-Surface to Ground Path Plan	
Step 1	Treat the grid model of the ground as water where neighboring cells with infinite traversibility costs are treated as an obstacle (see Figure 24).
Step 2	Use Algorithm to find the heading of the optimal path from the present location to the goal.
Step 3	Repeat Step 2 while the vehicle is in water.
Step 4	When the vehicle is firmly on ground, use Field D* to find the optimal path to the goal.

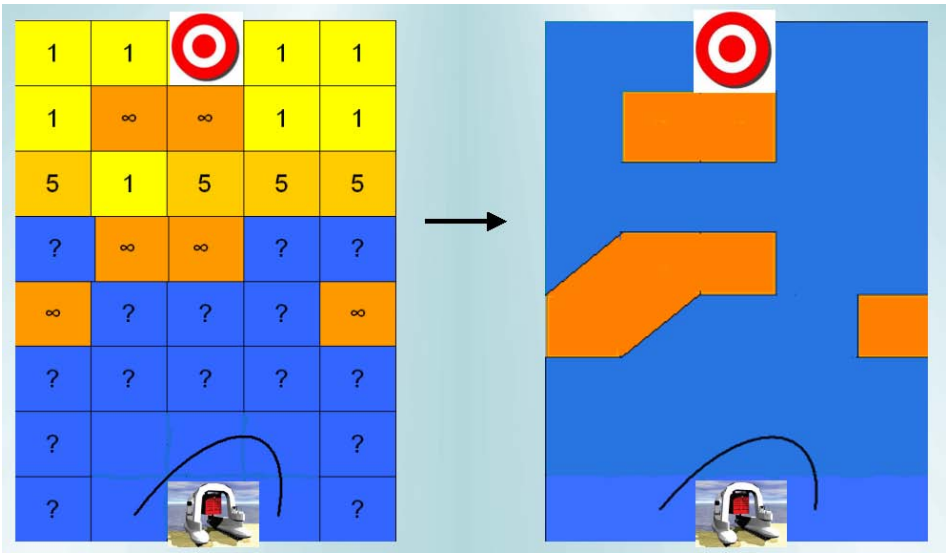

Figure 24: Illustration of Step 1 in Algorithm 11

Ground to Sea-Surface

Transitioning from ground to the sea poses less of a problem than the other way around because once the vehicle is in water, Algorithm can be used to reach the goal waypoint. Since the sea conditions change so rapidly, they should not be taken into account when

deriving the optimal path from ground to sea. The geometric configuration of the obstacles, however, should be taken into account which leads to Algorithm .

Algorithm 12: Computationally Feasible Ground to Sea-Surface Path Plan	
Step 1	Merge a grid representation of the sea, where obstacles represent neighboring cells with traversal costs of infinity and all other cells have a unit traversal cost, with the grid representation of land (see Figure 25).
Step 2	Use Field D* to find the optimal path from the present location to the goal location at sea.
Step 3	While the vehicle is on land, repeat Step 2.
Step 4	When the vehicle is in water, use Algorithm to find the optimal heading to reach the goal.

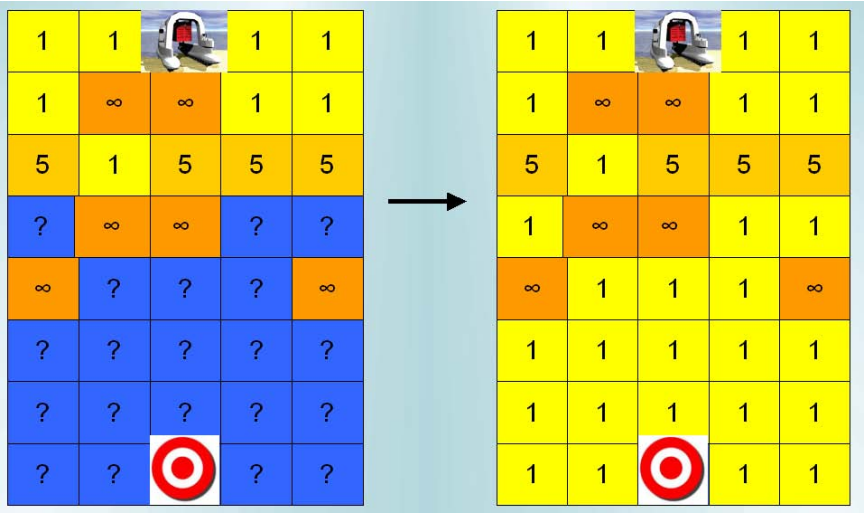

Figure 25: Illustration of Step 1 in Algorithm

Conclusion

Although AuAVs present a significant challenge in devising an optimized path-planning procedure, a number of options have been investigated and discussed and a potential algorithm for autonomous navigation has been identified and outlined:

- Existing theorems and algorithms for optimal sea paths were used to develop an original algorithm for continuous path planning for sea-to-sea navigation (Algorithm). The algorithm uses a simplified global path to derive an optimal heading the vehicle should take which has been shown to coincide with the heading of the actual optimal path (Dolinskaya & Smith, 2008b).
- An existing algorithm for autonomous ground navigation, Field D*, that is implemented on Mars Exploration Rovers was investigated and selected as feasible for this project (Algorithm).
- An original algorithm was proposed for the separate transition cases: sea-to-ground (Algorithm 11), and ground-to-sea (Algorithm). For sea-to-ground, the land area of operation is to be treated as sea, where cells with infinite traversal costs are treated as obstacles, until the vehicle beaches, at which point ground navigation (Algorithm) would be used to travel to the goal point. For ground-to-sea, the water area of operation is to be treated as ground, where cells either have costs of infinity (for obstacles), or one (lowest difficulty if there are no obstacles), until the vehicle enters the sea, at which point sea navigation (Algorithm) would be used to travel to the goal point.

Functional Requirements for Implementation

The proposed algorithms were developed as part of the design of an autonomous control system for DUKW-21. In order to successfully implement the navigation algorithms on DUKW-21 (or its current prototype – DUKW-ling), the assumed capabilities outlined below must also be developed, making them effective information requirements of the design.

- The AuAV will have access to a grid model of the global environment where each cell has an associated traversibility cost.
- The AuAV will have interoperability with other systems involved in the mission, which can provide updated information to the grid.
- There will be a means of creating a local map in real-time that can be efficiently integrated with the global map to create one grid.
- The amphibious vehicle will be able to use data it collects from onboard sensors to, in real-time, derive an approximate direction-dependent speed function that maps each heading angle to a maximum-attainable speed.

Providing these are met (with appropriate complexity depending on the application platform), the algorithm could be successfully implemented in either system.

Recommendations for Future Research

While the proposed algorithm finds an amphibious path that is feasible given environmental conditions, it does not factor in all of the vehicle operability constraints. In order to improve the accuracy of out path finding model, future development of this algorithm should consider turning constraints of the vehicle.

One ongoing research effort involves optimizing paths constrained by the sharpest feasible turns (Dolinskaya, 2009). However, this research does not address the presence of obstacles in the vessel domain. Alternatively, one can integrate a penalty, as a function of turn angle, which a vessel accrues every time it has to make a turn. This cost would be integrated into the ULPAR for sea navigation or the local cell costs for ground navigation. This way, the algorithm, for the most part, does not change; instead, the problem space is modified (which would change the heuristic used). Optimal motion of rotating non-circular robots is an ongoing field of research; a place to get started are the sections on optimal motion of non-point robots and multiple criteria optimal paths in Mitchell's survey (2000).

Additional work that must be completed in order to progress the development of an autonomous DUKW-21 include:

- Tests should be carried out on the remote-controlled DUKW-ling to better understand its dynamics in different sea states, as well as determine various constraints, such as maximum turning radius and the maximum slope gradient it is able to traverse.
- Research on Simultaneous Localization and Mapping (SLAM) algorithms to determine which ones are appropriate for DUKW-21's specifications.
- A hardware survey must be done to decide which sensors would be able to provide the necessary information for DUKW-21 to derive a direction-dependent speed function, as well as model the local environment to guarantee that the SLAM algorithm will be successful. The hardware must also be able to facilitate interoperability with other systems that will eventually be a part of DUKW-21's missions.

Bibliography

Carsten, Joseph. et al. (2006). "Global Planning on the Mars Exploration Rovers: Software Integration and Surface Testing," *Jet Propulsion Lab Technical Report.*

Chien, Steve. *et al.* (2000). "Using Iterative Repair to Improve the Responsiveness of Planning and Scheduling," *Proceedings of the Fifth International Conference on Artificial Intelligence Planning and Scheduling.*

Critchell, Lee. (2009). "DUKW-21 1:7 Scale Model Demonstrator 'DUKW-ling' Design Report," *Center for Innovation in Ship Design, Naval Surface Warfare Center, Carderock Division technical report.*

Dijkstra, Edsger. (1959) "A note on two problems in connexion with graphs," *Numerische Mathematik* 1:269-271.

Dolinskaya, Irina. (2009) "Optimal Path Finding in Direction, Location and Time Dependent Environments," *Ph.D. thesis (in preparation), University of Michigan.*

Dolinskaya, Irina. Smith, Robert. (2008a). "Fastest Path Planning for Direction Dependent Speed Functions," *University of Michigan technical report.*

Dolinskaya, Irina. Smith, Robert. (2008b). "Obstacle-Avoiding Fastest Paths in Anisotropic Media," *University of Michigan technical report.*

Ferguson, Dave. *et al.* (2005). "A Guide to Heuristic-based Path Planning," *In Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at The International Conference on Automated Planning and Scheduling (ICAPS).*

Ferguson, Dave. Stentz, Anthony. (2005). "The Field D* Algorithm for Improved Path Planning and Replanning in Uniform and Non-Uniform Cost Environments," *Carnegie Mellon University Technical Report*

Ferguson, Dave. Stentz, Anthony. (2006a). "Using interpoloation to improve path planning: The Field D* algorithm," *Journal of Field Robotics, Vol. 23.*

Ferguson, Dave. Stentz, Anthony. (2006b). "Multi-resolution Field D*," *Proceedings of the International conference on Intelligent Autonomous Systems*.

Gonzalez, Franklin. *et al.* (2007). "DUKW 21 – Amphibious cargo transfer from ship to shore," *Center for Innovation in Ship Design, Naval*

Surface Warfare Center, Carderock Division technical report.

Graham, R.L. (1972). "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1.

Hart, Peter E. *et al.* (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths," *IEEE Transactions on Systems Science and Cybernetics*.

Huntsberger, Terry. *et al.* (2006). "Intelligent Autonomy for Unmanned Sea Surface and Underwater Vehicles," *AUVSI Unmanned Science Newsletter*.

Koenig, Sven. Likhachev, Maxim. "D* Lite," In *Proceedings of the National Conference on Artificial Intelligence (AAAI),* 2002.

Madsen, P.A. *et al.* (1997). "Surf zone dynamics simulated by a Boussinesq type model. Part I. Model description and cross-shore motion of regular waves," *Coastal Engineering* Vol. 32.

Mitchell, Joseph S. (2000). "Geometric Shortest Paths and Network Optimization," *Handbook of Computational Geometry* Chapter 15.

Pell, Barney. *et al.* (1997). "An Autonomous Spacecraft Agent Prototype," *Agents Conference Proceedings*.

Rabideau, Gregg. *et al.* (1999). "Iterative Repair Planning for Spacecraft Operations Using the Aspen System," *Jet Propulsion Lab Technical Report*.